Exercise set 2:

3.2 Describe the actions taken by a kernel to context-switch between processes.
Answer:
In general, the operating system must save the state of the currently running process and restore the state of the process scheduled to be run next. Saving the state of a process typically includes the values of all the CPU registers in addition to memory allocation. Context switches must also perform many architecture-specific operations, including flushing data and instruction caches.

3.5 Including the initial parent process, how many processes are created by the program shown in Figure Figure 3.32?
Answer:
16 processes are created. The program online includes `printf()` statements to better understand how many processes have been created.

3.7 Using the program in Figure Figure 3.34, identify the values of `pid` at lines A, B, C, and D. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)
Answer:
Answer: A = 0, B = 2603, C = 2603, D = 2600

4.3 Which of the following components of program state are shared across threads in a multithreaded process?

a. Register values
b. Heap memory
c. Global variables
d. Stack memory

Answer:
The threads of a multithreaded process share heap memory and global variables. Each thread has its separate set of register values and a separate stack.

4.4 Can a multithreaded solution using multiple user-level threads achieve better performance on a multiprocessor system than on a single processor system? Explain.
Answer:
A multithreaded system comprising of multiple user-level threads cannot make use of the different processors in a multiprocessor system simultaneously. The operating system sees only a single process and will not schedule the different threads of the process on separate processors. Consequently, there is no performance benefit associated with executing multiple user-level threads on a multiprocessor system.

4.5 In Chapter 3, we discussed Google's Chrome browser and its practice of opening each new website in a separate process. Would the same benefits have been achieved if instead Chrome had been designed to open each new website in a separate thread? Explain.

Answer:
No. The primary reason for opening each website in a separate process is that if a web application in one website crashes, only that renderer process is affected, and the browser process, as well as other renderer processes, are unaffected. Because multiple threads all belong to the same process, any thread that crashes would affect the entire process.

4.6 Is it possible to have concurrency but not parallelism? Explain.
Answer:
Yes. Concurrency means that more than one process or thread is progressing at the same time. However, it does not imply that the processes are running simultaneously. The scheduling of tasks allows for concurrency, but parallelism is supported only on systems with more than one processing core.

4.10 Consider the following code segment:

```
pid_t pid;

pid = fork();
if (pid == 0) { /* child process */
      fork();
      thread_create( . . .);
}
fork();
```

a. How many unique processes are created?
b. How many unique threads are created?

Answer:
There are six processes and two threads.

4.12 The program shown in Figure 4.16 uses the Pthreads API. What would be the output from the program at `LINE C` and `LINE P`?
Answer:
Output at `LINE C` is 5. Output at `LINE P` is 0.

4.13 Consider a multiprocessor system and a multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be more than the number of processors in the system. Discuss the performance implications of the following scenarios.

a. The number of kernel threads allocated to the program is less than the number of processors.
b. The number of kernel threads allocated to the program is equal to the number of processors.
c. The number of kernel threads allocated to the program is greater than the number of processors but less than the number of user-level threads.

Answer:
When the number of kernel threads is less than the number of processors, then some of the processors would remain idle since the scheduler maps only kernel threads to processors and not user-level threads to processors. When the number of kernel threads is exactly equal to the number of processors, then it is possible that all of the processors might be utilized simultaneously. However, when a kernel-thread blocks inside the kernel (due to a page fault or while invoking system calls), the corresponding processor would remain idle. When there are more kernel threads than processors, a blocked kernel thread could be swapped out in favor of another kernel thread that is ready to execute, thereby increasing the utilization of the multiprocessor system.